# **Переменные**

|  |  |
| --- | --- |
| var a : Int = 7 | Изменяемая переменная |
| val a : Int = 7 | Неизменяемая |
| var b = 7 | Указывается либо тип, либо начальное значение т.к. Kotlin строготипизированных языков |
|  | Все типы – Объекты |
| toInt() toString() и т.п. | Преобразование типов |
| println (“age: ${a}”) | Подстановка переменных ${} |
|  |  |
|  |  |

# **Потоки ввода/вывода**

|  |  |
| --- | --- |
| var a = readLine() | Считать ввод с клавиатуры |

# **Функции**

|  |  |
| --- | --- |
| fun sum (a:Int = 0, b:Int = 2):Int {  return a+b  } | Объявление функции принимающей 2 int переменных и возвращает int |

# **Ветвления**

|  |  |
| --- | --- |
| when(a){  5 -> *print*("5")  6,7 -> *print*("6,7")  in 1..10 -> *print*("1..10")  in 0..200 -> {  *print*("0..200")  *print*("????")  }  else -> *print*("nothing") } | Похож на оператор switch. В отличии от java последнее условие не сработает. |
| when{  year == 1970 -> return("equals")  } |  |

# **Циклы**

|  |  |
| --- | --- |
| do, while | Как в java |
| for(i in 1..5){ | От 1 до 5 |
| for(cat in cats){  println("Кот $cat")  } | Из массива строк |
| val cats = arrayListOf("Barsik", "Murzik", "Vaska")  for( (index, element) in cats.withIndex()){  println("$index: $element")  } | Индексы и значения |
| for(i in 10 downTo 1 step 2){  print("$i ")  } | Итерация в обратную сторону от 10, с шагом 2 |
| for(i in 0 until cats.size){ | Замена стандартному for I=0;i<len;i++ |
| val daysOfWeek = listOf("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday")  daysOfWeek.forEach{  println(it)  } | forEach |
| repeat(3){  println("Васька! Иди жрать!")  } | Встроенная функция для повторения команд заданное число раз. |

# **Массивы**

|  |  |
| --- | --- |
| Для работы с массивами данных в kotlin существует три семейства контейнеров:   * Массивы - контейнеры фиксированной длины:   + не примитивных типов(Array),   + примитивных типов (\*Array - IntArray, CharArray ...), * Списки (\*List<T>) - контейнеры переменной длины. | |
| val array1 : Array<Int> = emptyArray() // пустой массив  val array2 : Array<Int> = arrayOf(1,2,3,4) // массив из элементов 1,2,3,4  val array3 : Array<Int> = Array<Int>(10, {i -> i\*i}) // массив из 10 элементов - квадратов 0-9 | Инициализация списков и массивов очень похожа. Для примера ниже опишем способы инициализации для массива целых чисел: |
| val list1 : List<Int> = emptyList() // пустой список  val list2 : List<Int> = listOf(1,2,3,4) // список из элементов 1,2,3,4  val list3 : List<Int> = List<Int>(10, {i -> i\*i}) // список из 10 элементов - квадратов чисел от 0 до 9 (будет создан ArrayList)  val list4 : MutableList<Int> = MutableList<Int>(10, {i -> i\*i}) // список из 10 элементов - квадратов чисел от 0 до 9 (будет создан ArrayList) | Для списков ситуация аналогичная: |

# **Исключения**

|  |  |
| --- | --- |
| try {  // some code  }  catch (e: SomeException) {  // handler  }  finally {  // optional finally block  } | В языке **Kotlin** нет проверяемых исключений. Из-за избыточности try в коде. В остальном как в java |
| throw MyException("Hi There!") | Для того, чтобы возбудить исключение явным образом, |
| val a: Int? = try { parseInt(input) } catch (e: NumberFormatException) { null } | Ключевое слово try является выражением, то есть оно может иметь возвращаемое значение. |
| class CatException() : Exception() | Собственный класс исключения |

# **Лямбды**

|  |  |
| --- | --- |
| { arguments -> function body } | Общий вид выражения. |
| {x: Int -> x + 5} | Открыли фигурную скобку, записали параметр в виде числа **Int**, а затем указали, что с ним нужно делать в правой части - прибавить к числу 5 |
|  |  |
|  |  |

# Nullability

| var a: Int = null // Ошибка компиляции  var b: Int? = null | Переменные которые могут быть null помечают «?», тогда компилятор не будет проверять. |
| --- | --- |
| var b = a?.length  //Что эквивалентно короткой  If (a == null){null}else{a. length } | Если мы хотим оставить проверку. Мы должны делать проверку на null через if != null  Есть еще кратная запись которая присвоит переменной значение null |
| var b=(a?. length) ?: -1 | Также можно добавить default значение, если null не устраивает, добавив элвис оператор «?:» |
| var b = a!!. length | Если же очень хочется nullPointerExeption. Добавляем «!!» |
| demo?.let { it.coolFun() }  //эквивалентно  if (demo != null ) demo.coolFun() | Помимо прочего, существует упрощенная запись проверки на null при помощи оператора let: |

# **Standard.kt**

|  |  |
| --- | --- |
| DbConnection.getConnection().let { connection ->  }  // connection здесь не доступен | **let()**Позволяет сокращать область видимости переменных |
| var demo : String? = null  demo?.let {  // если demo == null, этот блок не будет выполнен  } | Может использоваться в качестве проверки на null: |
| File(path).apply { mkdirs() } | **apply()** - это функция-расширение, которая выполняет лямбда-выражение аргумента и возвращает объект, над которым производилось действие.  Например:  Функция, которая получает в качестве аргумента String - путь к файлу, создает объект File, вызывает метод mkdirs() и возвращает File: |
| val prop = Properties()  FileInputStream("config.properties").use {      prop.load(it)  }  // FileInputStream автоматически closed() | **use()** - это аналог [**try-with-resources**](https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html)из Java. Эта функция может быть применена к объектам  Closeable, чтобы  автоматически закрывать объект. |

# **Полезные функции**

|  |  |
| --- | --- |
| val rnds = (0..10).random() | Рандом |
|  |  |
|  |  |

# **Draft**

# **Android**

|  |  |
| --- | --- |
| import android.util.Log  Log.d("Tag", "text") | Логгирование. Пример лога уровня дебаг. |
| **var** tw: TextView = findViewById(R.id.*textView*)  tw.*text* = i | Явный способ привязать переменную к id activity |
| **import** kotlinx.android.synthetic.main.activity\_main.\* | Чтобы не плодить эти вызовы. Можно импортировать все id, имена переменных будут == id |
| **var** tw: TextView = findViewById(R.id.*textView*) **var** bt: Button = findViewById(R.id.*button*) **var** clickCount = 0 bt.setOnClickListener **{** tw.setText(**"Click "** + clickCount)  clickCount++ **}** | Простой пример обработки нажатия кнопки. В методе onCreate |
| **fun** onCreate {  button.setOnClickListener(::buttonHandler)  **..**  **}**  **fun** buttonHandler(v: View) {  Log.d(**""**,**"clicked!"**) } | Вынос в отдельный метод в этом случае указывается метод обработки в setOnClickListener |

## draft

## draft

## draft

## draft

## draft

## draft

## Основные атрибуты у объектов View

* id - целочисленный идентификатор, который служит для обозначения уникальности объекта View в иерархии
* layout\_width - ширина объекта
* layout\_height -  высота объекта
* maxLength/minLength - максимальное/минимальное значение длины объекта
* maxWidth/minWidth - максимальное/минимальное значение ширины объекта
* layout\_margin -атрибут, который используется для установки внешних отступов
* constraints - это линии, на основе которых располагается view внутри ConstraintLayout.
* padding -  атрибут, который используется для установки внутренних отступов
* theme - стиль, который относится ко всему экрану активности или приложению, а не к отдельному компоненту приложения
* elevation - высота по координате Z
* background - фон объекта
* orientation - ориентация объекта (горизонтальная/вертикальная)
* rotation - значение угла поворота
* visibility - видимость объекта
* onClick - обработчик нажатия на объект
* scaleX/scaleY - изменить масштаб по X/ по Y